Specifying and verifying active vision-based robotic systems with the Signal environment

E. Marchand, Eric Rutten, Hervé Marchand, François Chaumette

The international Journal of Robotics Research,
Vol. 17, No. 4, April 1998, pp. 418-432

Special Issue on *Integrated Architecture for Robot Control and Programming*

Specifying and verifying active vision-based robotic systems with the SIGNAL environment

Éric Marchand, Éric Rutten
Hervé Marchand, François Chaumette

IRISA / INRIA Rennes
[1] EP-ATR project - [2] VISTA project
Campus de Beaulieu
35042 Rennes Cedex
E-mail: `Firstname.Name@irisa.fr`

Abstract

Active vision-based robot design involves a variety of techniques and formalisms, from kinematics to control theory, signal processing and computer science. The programming of such systems therefore requires environments with many different functionalities, in a very integrated fashion in order to ensure consistency of the different parts. In significant applications, the correct specification of the global controller is not simple to achieve, as it mixes different levels of behavior, and must respect properties. In this paper we want to advocate the use of a strongly integrated environment able to deal with the design of such systems from the specification of both continuous and discrete parts down to the verification of dynamic behavior. The synchronous language SIGNAL *is used here as a candidate integrated environment for the design of active vision systems. Our experiments show that* SIGNAL*, while not being an environment devoted to for robotics (but more generally dedicated to control theory and signal processing), presents functionalities and a degree of integration that are relevant to the safe design of active vision-based robotics system.*

1 Introduction/Motivation

The task of designing robot systems based on visual perception is made intrinsically complex by the very diversity of the problems that arise. Indeed, they cover a wide range of techniques and formalisms, including among others: mechanics, kinematics, electronics, signal processing, control theory, computer science, and discrete events systems. Each of these aspects is tipically handled

using dedicated tools that rely on various formalisms. One of the main issues is then the integration of these different aspects as well, as the underlying models of the actual tools. The use of a set of independent (or even loosely coupled) tools leads to the absence of any formal support for the integration. Modules are developed separately and then manually linked in an error-prone way. While the underlying models should be consistent, these tools perform different analysis and transformations using their own semantics. Our claim is that the concrete representations used by the tools should be as close as possible, or even, if possible, share a common format. This enables efficient and safe communication between tools, hence, smooth integration of functionalities, and even a formal verification of the whole system.

In this paper we propose a framework that features some functionalities for the development of active vision-based robotics systems. These systems contendwith various issues from the automatic generation of camera motion using image-based visual servoing to sensor planning. Each sub-problem involved in this kind of application (such as visual servoing, structure estimation, motion detection or segmentation, and exploration) is a difficult computer vision problem on its own. However, one of the main issues is the integration of all these tasks into a single, reliable, robust and safe autonomous system. We want to emphasize the fact that, if it is important to bridge the gap between continuous/local and discrete/global aspects in the vision and control parts of an active vision system, it is also important to consider this gap from a software engineering point of view to obtain a safe integration of such systems. Therefore, it is important to use a design environment that is able to provide tools that allow us to consider in a unified framework the various aspects of the perception-action cycle: from continuous data-flow tasks (or *sampled-data systems* which include control loops, estimation, filtering, and data acquisition) to multi-tasking and hierarchical task preemption (mission control). Furthermore, owing to the complexity of the system, and for safety requirements, formal verification tools are necessary to prove formally that the behavior resulting from the implementation suits the specification and is correct with respect to vision, robotics and control considerations.

Classical asynchronous programming languages are not really adapted to specifying and programming both the control and the mission levels of these systems. Furthermore they often do not rely on a formal model that supports verification. Therefore, our claim is that synchronous languages are better qualified to deal with a full integration. Our approach, is based on the SIGNAL environment. We propose a framework to specify and implement both the control level and the mission level using this environment. Our goal is not to provide a full description of the approach but we will try here to convince the reader that the integration of the different levels is necessary, and that synchronous languages in general and SIGNAL in particular are well qualified to answer the requirements of this kind of systems.

In the remainder of the paper, Section 2 describes more precisely what can be expected from an integrated design environment for vision-based robotic systems. In Section 3 the methodology we propose is briefly presented with regard to the main issues raised by such systems. As an example, we have applied the integration framework presented in this paper to the problem of scene reconstruction and exploration using a camera mounted on the end effector of a robot manipulator (Marchand and Chaumette 1997). We will illustrate the proposed integration framework in regard to the specification and verification of this application in Section 4.

2 Integrated Design Environments for Active Vision-Based Robotic Systems

2.1 Issues in Architecture Design

2.1.1 Real-Time Reactive Systems

We are interested here in the specification of reactive systems as defined in (Harel and Pnueli 1985). A reactive system can be divided into two parts: the environment and a sub-system which is the automated part of the system. The behavior of the latter is directly related to the informations acquired in the environment. The consequences of this definition are that such systems must be deterministic, they are subject to temporal constraints (to react to the pace of the environment), they must support parallelism (simultaneous arrival of information) and, finally, they must be reliable (we must be able to prove that their behavior suits the specification):

- **Determinism.** Determinism is fundamental for such systems. Indeed, the same flow of input data must produce the same control output and behavior. If such a basic property is not guaranteed, validation and certification of the system is not possible.
- **Parallelism.** A reactive system is a parallel system. Indeed, inputs acquired from the environment may come in simultaneously and may interest various parts of the system. Different processes must be able to receive *stimuli* from different sensors and must be able to react to all these promptings while following parallel evolutions. However these processes, with independent "lives", must be able to communicate with each other and share the informations acquired in the environment and the output responses of other processes. A strong synchronization framework has thus to be defined.
- **Reliability.** Reliability is a fundamental characteristic of reactive systems. The implemented system must be proved correct with respect to the specification. This means that we must verify liveness properties such as absence of deadlocks or data dependency cycles, as well as dynamic properties including *safety*, *liveness*, *reachability* and *attractivity*. Informally, whereas a liveness property stipulates that some *"good things"* do happen at some point during execution of the program, a safety property stipulates that some *"bad things"* do not happen at any point in any execution (Alpern and Schneider 1986).

2.1.2 Vision-based robot systems

The previous characteristics (determinism, parallelism and reliability) are the main requirements necessary to define a reactive system. In our case, we want to deal with the specific case of active vision systems. For such reactive systems, the design involves some other characteristics. Although information provided by a vision process may be simple enough to perform reflex actions, it is, unlike "basic" sensors such as proximity or force sensors, able to provide high-level information. This rich information (from the position of features in an image to 3D reconstruction, motion analysis or pattern recognition) may be used to build very complex and complete systems. However, despite the richness and diversity of the data it can provide, the major shortcomings which limit the performance of a vision system are usually its sensitivity to noise, its low accuracy, and its lack of reactivity. Therefore, in an active vision system, information extracted from the images are used in a purposive way for adaptively setting camera parameters (position, velocity,...) to improve the perception task.

The main characteristics of these systems is that data is extracted from images acquired at a regular pace (usually video rate) and that control outputs are sent to the actuator at the same pace. However, in any significant application, the design of an active vision-based robot system does not require a unique and low-level perception-action cycle. In most cases, it involves the definition of many different tasks. At a high level, a mission controller has to be designed. It must fully describe the goal to be achieved and reflect the strategy to be followed. This controller manages a set of subtasks which can be either sub-mission controllers or elementary tasks (usually a closed loop with respect to visual data, *e.g.*, a visual servoing process). Outputs of each elementary task are merged as directed by the controller in order to produce the final control which is sent to the actuators. The final step is to close the loop by acquiring a new image according to the new robot/camera parameters.

These systems can usually be divided into three different levels: a continuous and a discrete level (or control and task/mission levels), and a planning level. Therefore we must consider many different techniques. The sampled/continuous aspects (signal processing, control theory) rely on models that are different from those handling the aspects related to task-level control (computer science, discrete events systems). The planning or decision level should be a level on its own; however few systems feature this functionality and it is often integrated in the task level.

Let us sum up the requirements for the design and integration of an active vision system. As a reactive system, the underlying model must be deterministic and must support parallelism and verification. Data is obtained through a stream of images acquired at regular pace, therefore a data-flow model seems more appropriate than a classical imperative model. We must be able to express both the control level and the task level within the same framework. Each level has its own requirements:

- The design of the control and signal processing part involves functionalities like matrix calculations, simulators, symbolic calculus tools, algorithmic design assistance. Numerical libraries, mathematical packages, symbolic manipulation environments provide control engineers with assistance in the design of complex control laws, with tools for the analysis of these continuous regulation algorithms, and possibly automated code generation for implementating them.

- Concerning the task control aspects: the area of discrete events systems (*e.g.*, Ramadge and Wonham (1989)), the objects to be manipulated and constructed are mainly based on automata-like models. The model must support sequencing, preemption and parallelism between tasks. Other functionalities arise: specification of complex state-based behaviors possibly featuring interruption levels, performance evaluation, validation and/or verification, analysis of behaviors, code generation dedicated to the actual architecture, and software and hardware integration issues.

- The decision level is usually a planner that uses the current knowledge on the environment as well as the internal state of the robotic system to provide a succession of actions to be executed. The integration of this level within the architecture may be complex as the formalism used in the planner is very task-dependent. Therefore it should be interfaced with the two other levels using a format understandable by each part of the system.

The design of these systems is intrinsically complex, by the very diversity of the functionalities involved and the richness of the information acquired. Therefore it is important to consider an integration to be as complete as possible of the different parts of the system. This means that the design environment must feature coherent and communicating tools and not only a set of

tools where transfer from one to the other requires transformations and adaptations. In the latter case, this task has often to be achieved by hand, whereas, as one of the most complex and hence error-prone task, it would especially need assistance.

2.2 Related Work

This section briefly presents approaches related to ours, to position our results with regard to some relevant work in the field.

2.2.1 High Level Languages.

The most common model of time for concurrent programming is asynchrony (e.g., ADA, CSP, OCCAM or real time Operating Systems). The first approach is based on the expression of concurrency. Concurrent programming languages such as OCCAM (CSP) or ADA have numerous advantages. They are well structured and allow for good modularity. However, they are asynchronous and thus non deterministic. The synchronization between processes is performed during the execution and is unpredictable, thus they can hardly be used for reactive system implementation. The second approach relies on the connection of classical programs using real-time OS primitives. Here, the main problem is the number of programs to analyze and connect: diagnostic and maintenance are difficult, temporal constraints are not expressed in the program description but are satisfied using the OS primitives for processes synchronization and communication. This leads to systems which are generally non deterministic, and on which no safety properties can be formally guaranteed.

These approaches cannot be considered as integrated architectures. They do present functionalities relevant to the conception of an architecture for robot control. However they cannot handle the integration of continuous and discrete aspects of these systems and they do not feature validation tools. In contrast, modern high-level specification languages are based on sound formal bases, which support analysis tools (*e.g.*, analysis of data dependencies between computing processes, or their dynamical behaviors), as well as automated transformations for the optimization, verification, performance evaluation, distribution or automatic code generation from the specifications.

2.2.2 Dedicated Formalisms and Languages.

At this level, one can find that different paradigms of programming languages are better suited to different aspects of the system. For instance, on the one hand, the data-flow paradigm and style (and the associated block-diagram graphical style) are well suited to the specification of (sampled) continuous input-output functions, possibly with filtering effects. However, programming the sequencing of computations can be quite intricate. On the other hand, this is precisely the goal of imperative or automata-based language. In particular, the presence of preemption primitives in some languages provides for a direct expression of the starting, suspending, resuming and stopping of processes.

Let us however point out a few formalisms that can be used for the integration of CACE (Computer Aided Control Engineering). Most of the formal approaches are based on the use of transition-based systems. The finite state automata are well known tools: both deterministic and efficient, they allow the formal verification of properties. However, the composition of little automata can yield to a very big one, often impossible to understand; furthermore a little change in the specification can provoke a deep transformation of the automaton. Finally, let us point out that the expression of parallelism and the preemption of tasks are not supported by this formalism.

Petri nets are often used for small applications and if they support concurrency, they do not support hierarchical design, and they are not deterministic.

Other related work can be found in the area of Discrete Events Systems (DES) which can be related to transition systems based approaches (Ramadge and Wonham 1989). Using such approaches in robot vision (Aloimonos, Rivlin and Huang 1993; Kŏseká, Christensen and Bajcsy 1995) allows the synthesis of complex systems from the definition of simple independent behaviors. Kŏseká, Christensen and Bajcsy (1995) propose to use DES to monitor the behavior of a mobile robot. Some simple tasks (such as navigation, obstacles avoidance, etc.) are defined by simple automata. These automata are then composed using the methodology proposed by Ramadge and Wonham (1989) which leads to the definition of complex systems. The use of the DES framework allows the verification of the system. However, only the supervisor design is done using this formalism and the design of control task is not taken into account.

Finally, let us point out that hybrid systems (*e.g.*, Alur et al. 1995) allow to merge the continuous level and the discrete level within the same framework. More precisely, it allows to consider variables that take their values in various domains (such as boolean and real for example). However, there is usually no effective theory of such systems owing to the underlying undecidability of some properties (such as the accessibility). Several systems propose some solutions but are very restrictive in dealing with the way that properties can be expressed (restriction to linear hybrid systems for instance).

2.2.3 Dedicated Environments.

Other approaches appear to be more relevant with respect to the integration issue of robotic systems. Among them we can find ControllShell developed at Stanford by Schneider, Chen and Pardo-Castelotte (1995), Kheops developed at LAAS (Medeiros, Chatila and Fleury 1996) and Orccad developed at INRIA (Simon et al. 1993, Coste-Manière et al. 1996).

A useful integration is that of the continuous and discrete levels, in that it alleviates for the need to manually link executable modules after compilation through system-level primitives, which is generally fastidious and error-prone. It also enables for the construction of models where the interaction between the data-flow (continuous) and sequencing (discrete) parts of the system are effectively represented, and hence can actually take part in the analysis in the compilation and/or verification process.

ControlShell The system ControlShell features a variety of levels. At the level of control actions, it allows for the construction of models of components (such as PIDs) and trajectory generation in a data-flow language, provided with a block-diagram editor, and where processes operate in a sampled-data-driven way. Event-driven aspects are handled using daemons, which monitor states and trigger actions upon state change. Then, sequencing actions corresponds to defining phases in a mission, and is done using finite state machines. This environment features the integration of data-flow and sequencing formalisms. It enables the use of formal models such as finite state machines. However, it is not clear how the model can be used for an analysis and verification.

Hilare **and the** Kheops **System** Research activities around the robot Hilare at Laas also consider several levels according to the degree of abstraction of reasoning (Medeiros, Chatila and Fleury 1996). Modules are the levels where perception, action and modeling of the environment are handled. At the execution monitoring level, automata handle conflicts between modules. There the rule-based system Kheops is used for the specification of behavior. The task level involves

action procedures in the language PRS which is comparable to colored Petri nets. At the highest decisional level, involving symbolic reasoning on actions to be taken with regards to objectives, a planning functionality is offered.

The ORCCAD System The system ORCCAD developed at INRIA also has a hierarchical organization, according to the frequency of interactions with the environment. It has a functional level, where the robot tasks are defined as control laws associated with start and stop. The control level is where robot procedures handle the reactive aspects of the behavior, in a way supporting verification. Finally, at the decision level, a planner could generate robot procedures in an exchange format. The research results around ORCCAD concerning our domain of interest feature performance evaluation using the SIMPARC simulator, static analysis and code generation using ESTEREL, verification of discrete-event behaviors using AUTO, and timing analysis using TIMED ARGOS and KRONOS (Espiau et al. 1995; Kapellos et al. 1995). They cover a wide range of the functionalities one would like to be provided with in a robotics design environment. The question is how integrated this is; the fact that the control level is not expressed in the orbit of the synchronous approach may be a problem.

These various environments do not feature the same functionalities which are summed up in Tables 1 and 2. We have presented what an integrated CACE should be and what solutions are usually adopted to specify such systems. We now describe our environment based on the synchronous language SIGNAL.

3 A Synchronous Integrated CACE based on SIGNAL

We examine how our approach can handle appropriately some of the problems which may arise from the specification of both data-flow continuous parts and discrete event parts towards code verification for safety purpose. Our architecture is divided into two main levels. The lowest level handles the dynamic and reactive aspects of the system while the highest level handles the definition of the mission and makes the decisions.

3.1 SIGNAL: an answer to our requirements

The technique involved for the integration is the *synchronous approach* to reactive real time systems (Berry 1989). One way of interpreting the synchrony hypothesis consists in considering that computations produce values that are relevant within a single logical instant of time. A family of languages is based on this hypothesis (Halbwachs 1993). They are all provided with environments featuring tools supporting specification, formal verification and generation of executable code, all based on their formal semantics. Among them, SIGNAL is a real-time synchronized data-flow language designed with a specific attention to the application domain of signal processing and control systems (Le Guernic et al. 1991). We advocate that SIGNAL is an environment that fulfills most of the requirements we have exposed in the previous section.

We have exhibited three different levels relevant in a vision-based robotic architecture. Let us first examine shortly how SIGNAL contends with theses various aspects:

- Dealing with the sampled continuous level, the data flow framework is particularly appropriate for the specification of active vision systems because of the equational and data flow nature of the closed-loop control laws, which can be implemented as control functions between

sensor data and control outputs. The possibility of implicitly specifying parallel processes is useful to compose behaviors. Finally, the synchrony hypothesis corresponds well to the model of time in the equations defining the control laws.

- The second point concerns tasks sequencing and preempting at the discrete level. The language-level integration of the data flow and sequencing frameworks can be achieved using an extension of SIGNAL: SIGNAL*GTi* (Rutten and Le Guernic. 1995). SIGNAL*GTi* enables the definition of time intervals, their association with data flow processes and provides constructs for the specification of *hierarchical preemptive tasks*. This way, it offers a multi-paradigm language combining the data flow and multi-tasking paradigms, for hybrid applications blending (sampled) continuous and discrete transition aspects. Using SIGNAL*GTi*, we can design a *hierarchy* of parallel automata, thus we have the advantages of both the automata (determinism, tasks sequencing) and concurrent programming languages (parallelism between tasks) without their drawbacks.

- The decision level is not currently featured in the SIGNAL environment but may be incorporated into the task level.

Integration is complete because the semantics of SIGNAL is defined via a mathematical model (based on the synchronous hypothesis). Both levels are described using this model. Describing this model is not the goal of this paper (and may remain unknown from the point of view of the robot programmer). Let us just say that a SIGNAL program describes relations between flows of data and events. The compiler transforms the program into a system of equations and then calculates the solutions of the system which may thus be used as a proof system. Its programming environment, which is not limited to the compiler, features tools for the automated analysis of formal properties. The compilation of SIGNAL code provides a dependencies graph on which static correctness proofs can be derived: it automatically checks the network of dependencies between data flows and detects causal cycles, temporal inconsistencies from the point of view of time indexes. SIGNAL automatically synthesizes the scheduling of the operations involved inside a control-loop (note that this work is an error-prone task when done by hand in classical C-like languages), and this scheduling is proved to be correct from the point of view of data dependencies. The SIGNAL code is thus easy to modify since the re-synthesis is automatic. Finally, the compiler synthesizes automatically a global optimization of the dependencies graph. Furthermore, SIGALI, the model checker, allows us to prove safety properties. All of these functionalities are integrated in the SIGNAL programming environment (see Figure 1) which is organized around the hierarchical synchronized data-flow graph. The functionalities listed above are all based on this representation. In that sense, the whole design process requires no manual transformation of models from one tool to the other. SIGNAL can be seen as a fully integrated environment.

We must mention that a data-flow language such as SIGNAL is not adapted to all kinds of computations. For example, image processing or linear algebra cannot generally be performed with SIGNAL (or, at least, not without difficulty). However, the use of such functions is not performed asynchronously: they are considered as any function defined in SIGNAL, thus we do not leave the synchronous framework. Furthermore, the management of asynchronous inputs or interruptions is not supported, but this is not necessary in vision-based applications where the inputs are provided regularly and periodically at video rate. Finally, dynamical management of time at the execution is also not treated here, but this is not necessary due to the regular aspect of the loops.

An other example of the integration of data-flow and sequencing languages within the synchronous framework is ARGOS and LUSTRE proposed by Jourdan et al. (1994).

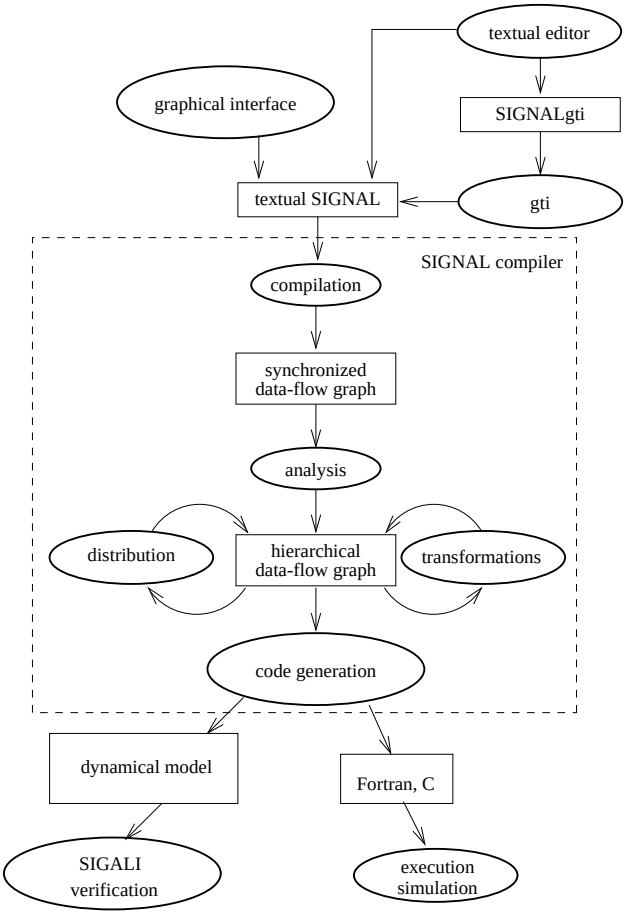

Figure 1: The SIGNAL environement

3.2 Data-flow processes for the continuous level

The general motivations for the application of a data flow language to robot control come from the following observations. A robot control law, at the relatively lowest level, consists in the regulation of a task function, which is an equation $c = f(s)$ giving the value of the control c to be applied to the actuators, in terms of the values s acquired by the sensors. The control of the actuator is a continuous function f, that can be complex. Such a task can be composed of several sub-tasks, with a priority order. The implementation of such a control law is made by sampling sensor information s into a flow of values s_t, which are used to compute the flow of commands c_t: $\forall t, c_t = f(s_t)$. This kind of numerical, data flow computation is the dedicated application domain of data flow languages in general, and of SIGNAL in particular. As indicated by the time index t in this schematical equation, the values involved are simultaneously present, and this is preserved when several such equations are composed.

The types of data handled are vectors and matrices of reals, and the operations performed are arithmetic, inversion, etc. The set of operations is to be performed on each input data that at each instant in this logical time. It corresponds to the control theory equations (given in continuous time) adapted to the discretization of sampled sensor values. The considered algorithms have two specific features. First, they have an equational nature: they express relations between various flows of data, in a declarative way. In particular, the iterative aspect in the control loop (at each

instant) is completely implicit. Second, they are synchronous: the equations involve values of the different quantities at the same logical instant.

Classical programming methods are not so well adapted in specifying and programming such algorithms; asynchronous imperative languages require the explicit management of low level aspects of the implementation (like the sequencing of computations imposed by data dependencies), and of the temporal aspects (e.g., down-samplings on a flow of data, multi-rate parallel computations), for which there is no well-founded support or model. On the other hand, the synchronous data flow language SIGNAL provides the adequate high level of abstraction for declarative specification, as well as a coherent and powerful model of time.

3.3 The Controller: Tasks on Nested Time Intervals

Once a library of control tasks has been built, the specification of higher-level and more complex behaviors requires the possibility to combine these tasks in various ways. Especially, one wants to combine them in sequence or in parallel, starting and interrupting them on the occurrence of events, which can be either external (coming from logical sensors) or internal (e.g., the reaching of certain thresholds). This level of robot programming necessitates preemption structures for concurrent tasks. The purpose of SIGNAL*GTi* (Rutten and Le Guernic 1995) is precisely to augment SIGNAL with objects and operations for the construction of such preemptive hierarchies of data flow tasks. Thus, preemption, parallelism and sequencing of data flow tasks is handled in an extension to SIGNAL using the notion of time interval. This enables the specification of hierarchical interruption structures (such as with the STATECHARTS (Harel 1987) or ARGOS (Jourdan et al. 1994)), associating data flow computations to execution intervals, and making transitions from the one to the other in reaction to events.

Once the specification is done, the different intervals have to be defined as must the corresponding process. Then, different kinds of behavior can be expressed. For example, according to the intervals defined in Figure 2, `P1 each I1` means that P1 will be executed when `I1` is active, `P1 each I1 | P2 each I2` means that `P1` and `P2` will be executed in sequence (because `I2` begins when `I1` ends), while `P1 each I1 | P2 each I1` means that they will be executed in parallel. The discrete events $\texttt{E}_\texttt{i}$ may be emitted either by the processes executed on these intervals or by any other process executed in parallel on another interval. Parallelism between processes is implicit and we do not have to take special care of the synchronization (which is handled automatically by the compiler).

3.4 Formal Verification for Safe Sesign

Like all the other synchronous languages, SIGNAL features some verification tools. Static properties (*e.g.*, the absence of cycle in the data dependencies) can be checked automatically at the compilation on the whole system. However, dealing with dynamical behavior, it is actually not possible to verify the continuous level of the application (other than in simulation, as with ORCCAD), but the mission level can be verified through the use of SIGALI. The reader interested in the theoretical foundation of the verification approach is referred to Le Borgne et. al (1989) and Le Borgne et. al (1996). The equational nature of the SIGNAL language makes it natural to use an equational framework for modeling discrete behaviors and proving properties on them. This description of dynamical systems using equations is quite common in the fields of control theory and digital circuits, but not in verification and model-checking. This aspect is an originality of the SIGNAL approach compared

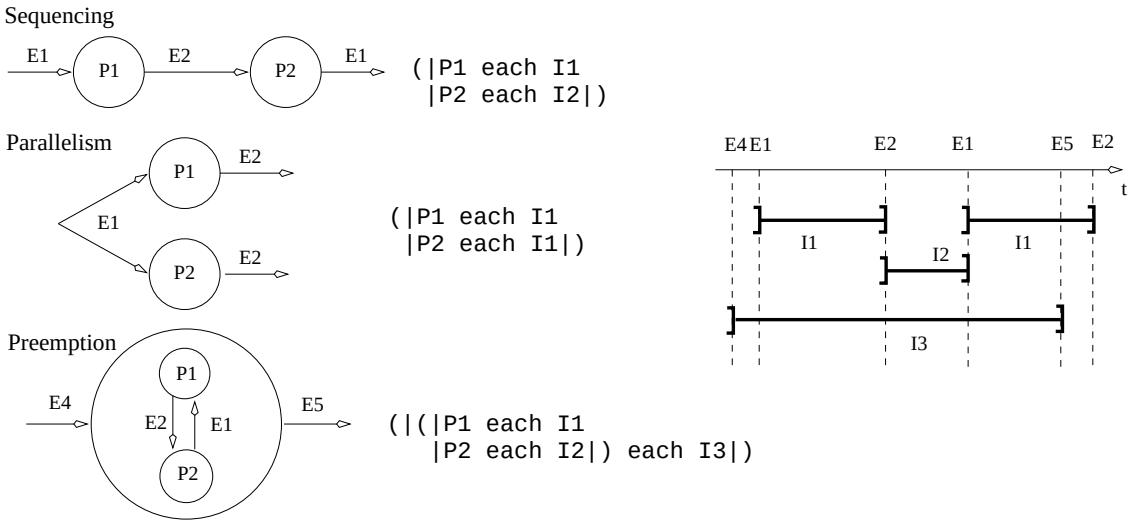

Figure 2: Example of sequencing, parallelism and preemption using time intervals

to others which use transition systems (for example, ESTEREL (Boussinot et. al 1991) and LUSTRE (Halbwachs et. al 1992)).

The formal basis of SIGNAL allows the verification of dynamic properties such that *safety*, *liveness*, *reachability* and *attractivity*. As already stated, informally, whereas a liveness property stipulates that some *"good things"* do happen at some point during any execution of the program, a safety property stipulates that some *"bad things"* do not happen at any point of any execution (Alpern et. al. 86). For example we can check that two processes can (or cannot) be active in the same time (reachability), that a given state will always be reached (attractivity), that the system cannot enter into a state if a given precondition has not previously been achieved (invariance). The SIGNAL program is translated in a set of equations on which the properties are checked. To be used easily by the end-user, each property to be checked can be expressed as a logical assertion on the system behavior.

The ability to verify formally the behavior of the system is a fundamental aspect of the proposed architecture. These properties (both static and dynamic) checking tools are important at two levels: for development purposes, it is important to verify that the system really has the expected or required behavior; and for the certification of the safety of the system, which is meaningful regarding safety-critical applications like most of the robot vision applications.

The contribution of the synchronous approach, and of SIGNAL in particular, is that it has a programming style closer to a control engineer's specification and that they provide the programmer with a set of tools that offer relief from error-prone tasks. The language-level integration of the data flow and task preemption paradigms enables the design of time intervals, their association with data flow processes in order to form tasks, and the sequencing of these data flow tasks. This way, the whole application can be specified *within the same framework and using the same underlying formal model* from the discrete event driven transition behavior down to the (sampled) continuous servoing loop.

4 An Application in Vision-Based Robotics

4.1 Overview of Our Application

We have applied our integration framework to the problem of scene exploration (Marchand et al. 1997a) using an eye-in-hand system composed of a calibrated camera mounted on the end effector of a 6 d.o.f. manipulator (see Figure 3). More precisely, we are interested in scenes made up of cylinders and polyhedral objects without any knowledge of their number, their location and of their dimensions. Our goal is to obtain a 3D map of the scene that is as accurate and as complete as possible. Only the global dimension of the scene (1 m^3 typically) is assumed to be known. Describing the whole application in details is not the goal of this paper. We concentrate here only on the aspects that are important from the point of view of programming and integration.

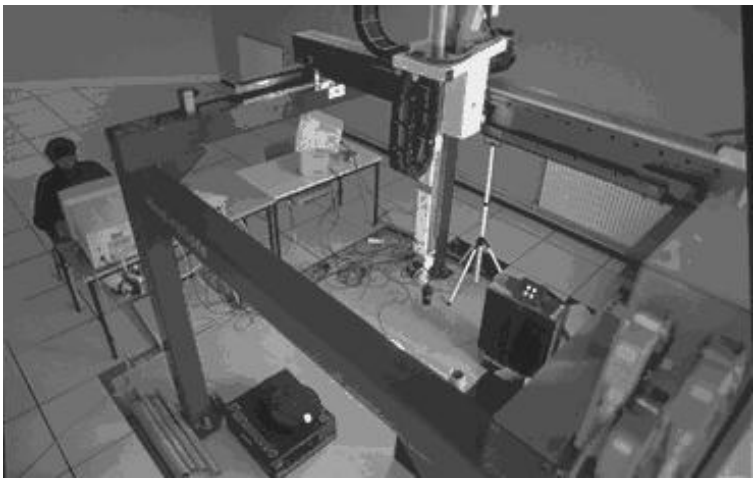

Figure 3: Eye-in-hand system involved in the experiments

The entire reconstruction/exploration process is roughly described using a hierarchical parallel automaton (see Figure 4). This system has three main perception-action cycles (*i.e.*, three levels).

4.1.1 The Exploration Cycle

The first one ends only when the reconstruction is as complete as possible. Its goal is to discover objects which have not been yet observed by the camera. It is based on perceptual strategies able to determine successive camera viewpoints that improve the observation of the scene. We have defined a gaze planning method which proposes a solution to the next best view problem. It mainly uses a representation of known and unknown areas as a basis for computing new viewpoints. The exploration cycle deals with global 3D informations and the resulting camera motions are discrete.

4.1.2 The Incremental Reconstruction Loop

When an object is observed, the system enters the second cycle which, is the incremental reconstruction loop. The main goal of this level is to bridge the gap between a local modeling of the scene and a global one. Indeed, at this level a reconstruction of all the objects observed from the current viewpoint is performed in order to obtain as accurate results as possible, we have chosen to perform the reconstruction in sequence. This is owing to the fact that the camera motion is controlled for

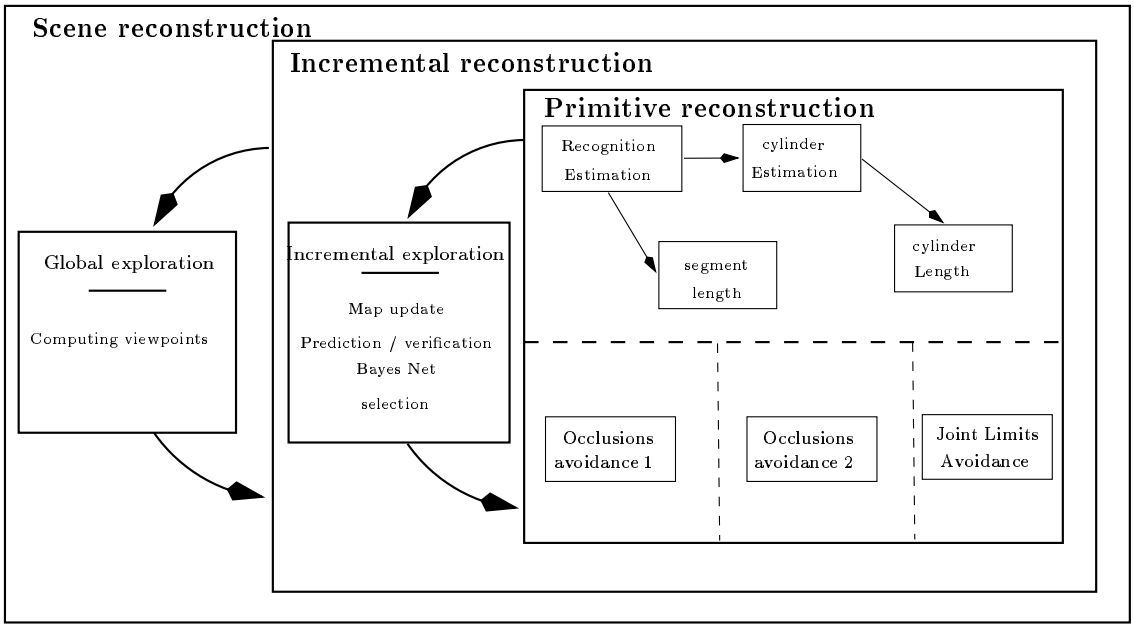

Figure 4: Hierarchical parallel automata describing the reconstruction/exploration process

the precise structure estimation of one primitive at a time (see the next cycle). Note that after the reconstruction of a primitive, the camera is located at a new position where new objects may be observed. We have thus developed a simple algorithm which allows to ensure that all the primitives observed from a set of camera positions have been reconstructed. This point is important for taking into account all the available information, and, above all, for correctly computing the known free space and unknown area involved in the global exploration process.

4.1.3 Active Reconstruction of a Primitive

The latter cycle deals with the active reconstruction of a primitive which is based on a local and continuous structure from motion approach (Chaumette et.al 1996). It consists in estimating the set of parameters $\mathbf{P}$ which describes the 3D position and dimension of a primitive $\mathcal{P}_i$ assumed to be represented by an equation of the type $h(\mathbf{X}, \mathbf{P}) = 0, \forall \mathbf{X} \in \mathcal{P}_i$. More precisely, we have:

$$\mathbf{P} = \mathbf{P}\left(\mathbf{p}, \dot{\mathbf{p}}, \mathbf{T}\right) \tag{1}$$

where $\mathbf{p}$ describes the position of the primitive in the image, $\dot{\mathbf{p}}$ is its velocity, and $\mathbf{T}$ is a measure of the corresponding camera velocity. Let us note that, when no particular strategy concerning camera motion is defined, important errors on the 3D structure estimation can be observed. This is owing to the fact that the quality of the estimation is very sensitive to the nature of the successive camera motions. An active vision paradigm is thus necessary to improve the accuracy of the estimation results by generating adequate camera motions. More precisely, it has been shown that the primitive must remain static at a particular position in the image during the camera motion in order to obtain a non-biased estimation robust with respect to measurement errors (see Figure 5 where the cases of a straight line and a cylinder (using its two limbs) is described). In practice, such gaze control task is realized by visual servoing (Espiau et. al 1992).

For each observed segment in the image, a recognition task is first performed in order to determine the nature of the corresponding 3D primitive (cylinder or segment). Then, if a cylinder

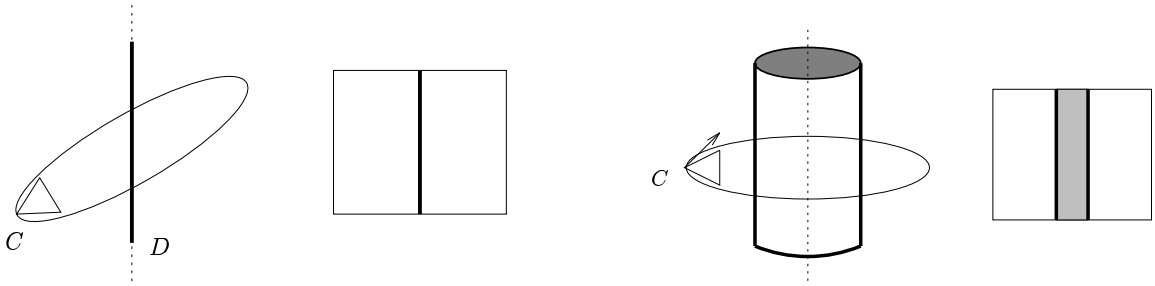

Figure 5: Optimal camera motion and resulting image in the cases of a straight line and a cylinder

has been recognized, an estimation of its parameters based on its two limbs is carried out in order to achieve a more robust reconstruction. Finally in both cases, the length of the primitive is computed. In parallel to the reconstruction tasks, due to the camera motion, occlusions and robot joint limits avoidance are realized.

At each level, the behavior of the robot is fully directed by the information acquired by the camera. However, the way data is processed and used is different. At the lowest level, images are acquired at video rate, and the camera motion is automatically generated by visual servoing, leading to a continuous behavior. At the highest level, only key images are acquired and the resulting interpretation will allow to determine a new camera viewpoint, leading here to a discrete behavior.

4.2 Specification in SIGNAL

Let us now examine how such an active vision-based application can be specified in the SIGNAL environment.

Continuous (sampled) level: Without giving too much details (see (Marchand et al. 1997b), from the specification/implementation point of view, the first step was to create a library of vision-based tasks that ate based on visual servoing. It contains the basic positioning tasks which use only image data and the following secondary tasks:

- joint limits and singularities avoidance (Marchand et al 1996) until the convergence of the gazing task. This step allows to attain a correct robot position before the execution of the following tasks; and
- trajectory tracking which allows the camera to move along a desired trajectory. This is used for the active reconstruction to turn around the selected object (see Figure 5). In that case, occlusion that may occur in the image and joint limits are avoided by a simple change of direction along the desired trajectory.

This library has been implemented in SIGNAL except image processing and some numerical computation (such as matrix inversion) which have been implemented in C and are called from the SIGNAL program. As already stated, once the execution of these functions respects the synchronous hypothesis, we do not leave the synchronous framework. Useful functionalities of SIGNAL at this level are the implicit closed-loop (owing to the data-flow nature of the language) and the implicit parallelism (*e.g.*, to consider a trajectory tracking process in parallel with a gazing task). Once this

library is available, estimation processes have been added "in parallel". Therefore an other library has been constructed which proposes functions for the 3D reconstruction of primitives (segments, cylinders using one or two limbs, ...) by dynamic vision. Here, we have used here functionalities such as the delay to have access to the past value of a signal (variable). At this level, everything is implemented in SIGNAL, from the estimation itself to filtering processes. As already stated (see Eq. 1), the structure estimation method is based on the measure of $\dot{\mathbf{p}}$, which is computed using the current and past values $\mathbf{p}_t$ and $\mathbf{p}_{t-1}$. It is also based on the measure of the camera velocity between these two instants t and $t-1$. In SIGNAL, the past value of $\mathbf{p}$ and of the camera velocity can be easily expressed using the delay operator `$`.

When these libraries are available, we compose an estimation process and the control algorithms which allow the automatic generation of the optimal trajectory and joint limits avoidance. Therefore, if `p` is a signal carrying the position of the primitive in the image and `Tm` the measure of the camera velocity, the computed camera velocity `Tc` and the estimated parameters `P` are expressed by :

```
(| P  := ESTIMATION{p,p$1,Tm$1}
 | Tc := gazing_task(p,P) + secondary_task
 |)
```

The integration of these two processes (*i.e.*, control and estimation) is interesting. It allows us to show that:

- the parallelism between these two tasks is implicit. The estimation is performed in *parallel* with the visual servoing task as expressed by the composition operator of SIGNAL seen here as a parallel operator ; and
- we do not have to build the closed loop using an iterative process: control equations can be written "as is" ; and
- access to past values of a variable, useful for filtering for instance, is featured by SIGNAL according to the data flow nature of the language. Set up temporary memory is not necessary ; and
- from the verification point of view, data dependencies and logical time coherence are automatically checked by the compiler: no cycle or dead-lock can occur.

The mission controller From the point of view of the mission controller, we have three levels: exploration, incremental reconstruction and primitive reconstruction. Each level can be represented by an automaton where the sub-level is nothing but a state. Therefore, we have defined *tasks* which associate a given process (*e.g.*, the structure estimation) with a *time interval* on which they are active. The transitions between tasks are discrete events. Using the notion of task (time interval+process) allows us to specify easily simple automaton and even hierarchical automata. Figure 6 gives a description of the resulting automaton. Figure 7 gives an example of the behavior of the system in terms of intervals.

Let us just give a few examples: the three main levels are specified using preemption (whose names correspond to those given on Figure 6 and 7) to go from one level to another. For example, after an exploration process (on interval `I_exploration`), the system enter the incremental exploration level only if a new primitive is observed in the environment (defined as `n\=0` on Figure 6 and 7). Inside the lowest level, we execute in sequence the recognition process (on interval

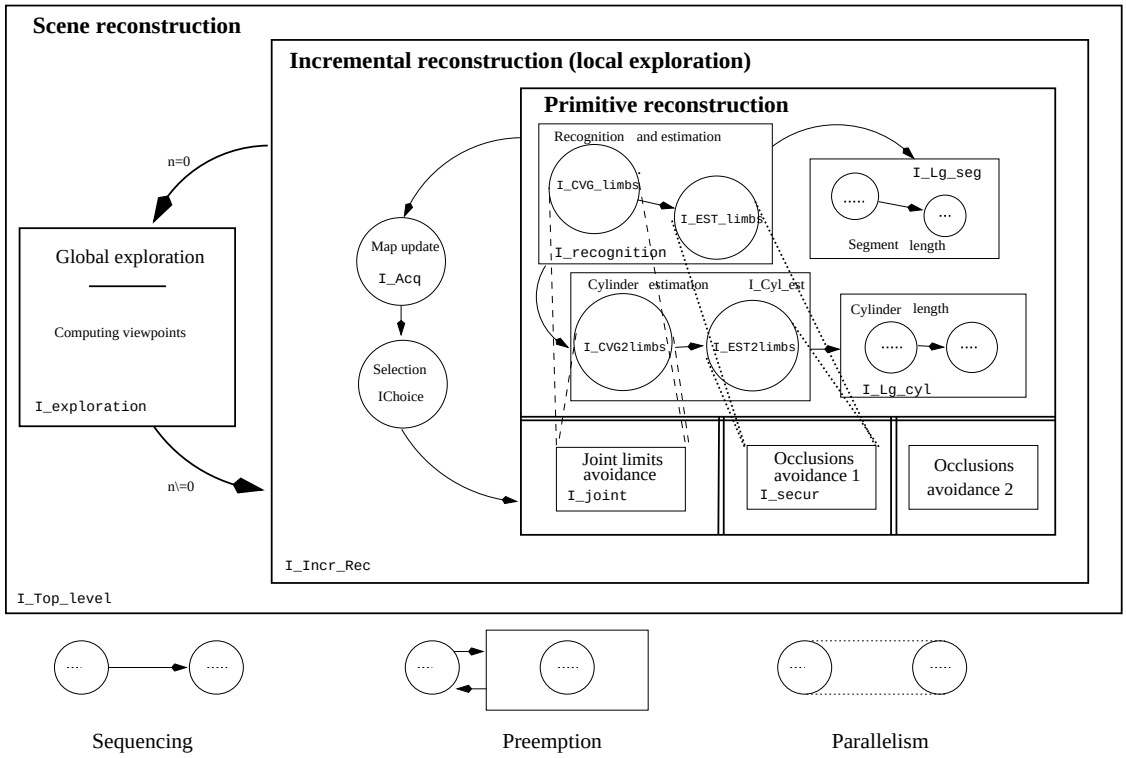

Figure 6: Hierarchical parallel automata describing the reconstruction/exploration process

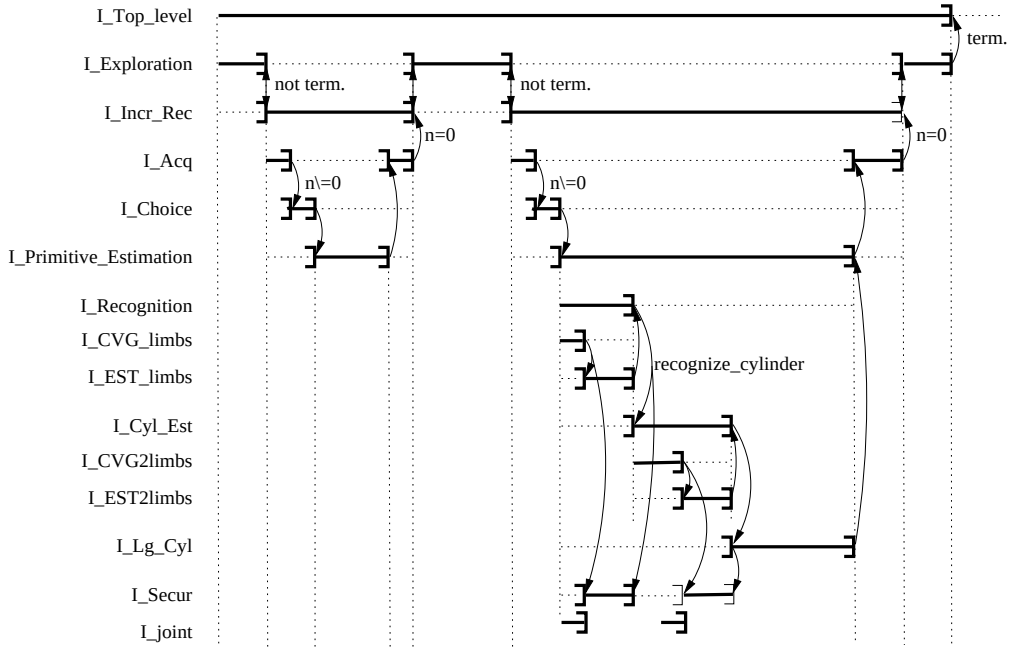

Figure 7: Specification of the sequencing in terms of activity intervals: a possible trace.

I_recognition), and either the reconstruction of a cylinder (each I_Cyl_est) or the estimation of a segment length (each I_Lg_seg). Each one of these processes is also decomposed into two parts, executed in sequence, corresponding to a visual servoing task and a parameter estimation task (for example I_CVG_limbs followed by I_EST_limbs). In parallel with this recognition and reconstruction task, an occlusion avoidance (I_secur) process is performed, as well as a joint limits avoidance (I_joint) process (only during the visual servoing). Such a description may appear complex. However, it is driven by the problem we have to solve. Let us just note that specifying such behavior in C is extremely difficult (especially when dealing with parallelism and preemption). Here, once the specification of this hierarchical parallel automaton is completed, its translation in terms of intervals and then the association between each process and the corresponding interval(s) is straightforward. Each interval is specified using the events which define its beginning and its end. At this level, parallelism between tasks as well as preemption have been widely used. Once the automaton is available, we have associated to each state one of the vision-based tasks which have been previously implemented or other tasks, such as the computation of the viewpoints in the exploration process, written in C++ and which are called as an external function. The resulting program is a SIGNAL program which integrates either the continuous parts (control and estimation) and the discrete part (mission controller).

The compiler is then used to generate a code in C. Next step is to validate the code with respect to complex properties using SIGALI.

Verification We have applied the verification tools available in the SIGNAL environment to check various properties of our implementation of the vision tasks and controller. We give here some examples of the properties which have been proved:

- inside a reconstruction task, the estimation and the joint limits avoidance processes cannot be active at the same time. This property has to be verified because the number of degrees of freedom of our robot cannot allow the simultaneous activation of both tasks. We have just to check that the two states (described by their time intervals) cannot be reached at the same time: *i.e.*, according to notation used on Figures 4 and 7, that `reachable(true(I_joint) and true(I_EST_limbs))` is always false which is automatically verified using SIGALI ;

- the joint limits and occlusion avoidances are active only when an estimation is performed. This property is more complex to check since the estimation process is handled in two different tasks. Furthermore, these processes, as described in the previous paragraph, are not in the same layer in the hierarchical automata ;

- the reconstruction will end, that is the final state will be reached. Let us note that this proof is correct with respect to the fact that the volume of observed area cannot decrease. This assumption, obviously true, can be added to the proof system without changing the behavior of the global system ;

- when a primitive is observed, it necessarily implies that a recognition process followed by either a cylinder estimation or a length segment estimation is performed.

These properties are just examples, more examples can be proposed. Let us note that knowledge on the underlying method used in SIGALI is not required. Indeed, the model required for the modeling of the system and the properties specification is unified. However a good understanding of what these properties mean is necessary in order to translate them into a logical assertion. Note that dealing with invariant properties (*i.e.*, true for all instants such that there are no deadlock

or cycles in the program), they are automatically checked during the compilation of the program. When the executable code is provided, it is proved to be deadlock free.

5 Conclusion

We have presented a framework that allows us to produce a safe design of active vision-based robotics applications within the synchronous design environment SIGNAL. The SIGNAL environment is not specialized for robotics applications, but more generally concerns the specification of reactive systems. However its degree of integration between the different tools (specification, compilation, verification, code generation) is an answer to the requirements in sensor-based robotics systems design and to active vision systems in particular. Hence we claim that, dealing with our integration goal, the synchronous data-flow approach is advantageous for three main reasons:

- the programming style we have proposed for this kind of application is independent from the sequential aspect of the computer architecture, and thus close to the original specification of the control theorist ;
- it allows us to consider in **unified framework** the various aspects of the application: from data-flow task specification to tasking and hierarchical task preemption ; and
- dealing with such systems, the tool for verifying properties is important at two levels: for development purposes, it is important to verify that the system really has the expected or required behavior ; and for the **certification of the safety** of the system, which is meaningful regarding safety-critical application.

Obviously, the proposed environment does not feature all the requirements we have exhibited. In particular, the decision level is not featured (automatic synthesis of controllers using optimal control theory may be one answer to this problem (Marchand and Le Borgne 1997). There is also no way to verify the continuous level. Indeed, SIGNAL does not study a continuous signal but only sampled data (like CONTROLSHELL). Therefore we can check only the temporal coherence of data. However, what can be done within this framework is done in a fully integrated way, using the same underlying formal model. Furthermore, most of the error-prone tasks (such as parallelism specification, closed-loop, composition of behavior, etc) are achieved automatically and efficiently thanks to the same model. Having a unified model for both the data-flow and tasking aspects does however raise the question whether the cost in complexity of the models is viable. Bigger applications, some of them bigger than ours in term of number of states, have been specified and implemented using the same framework. Each time the compiler manages to optimize the internal representation of the program (*i.e.*, the synchronized data-flow graph on Figure 1). Therefore the number of states does not increase widely with the size of the application. Finally, the interest of this approach has been validated with real time experiments devoted to structure estimation and exploration of static scenes. We are convinced that many other active vision-based applications, such as target tracking or visual survey, can be easily implemented using the same architecture framework.

Acknowledgment

This work was partly supported by the MESR (French Ministry of the University and Research) within project VIA (*Vision Intentionnelle et Action*) and under contribution to a student grant.

Authors want to thank the anonymous reviewers and the guest editors for their remarks.

Refererences

Aloimonos, Y., Rivlin, E., and Huang, L. 1993. *Designing Visual Systems: Purposive Navigation,* In Active Perception*, Aloimonos Y. Editor*, pages 47–102. Lawrence Erlbaum Assoc. publishers, Hillsdale, New Jersey.

Alpern, B., and Schneider, F.B. 1986. Recognizing safety and liveness. Technical Report 86-727, Departement of Computer Science Cornell University, Ithaca, New York.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H. Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. 1995. *The Algorithmic Analysis of Hybrid Systems*, Theorical Computer Science, 138:3-34

Boussinot, F., and De Simone, R. 1991. The ESTEREL language. *Proceedings of the IEEE*, 9(79):1293–1304.

Berry, G. 1989. Real time programming: Special purpose languages or general purpose languages. *11th IFIP World Congress*, San Francisco, California, pp. 11–17.

Chaumette, F., Boukir, S., Bouthemy, P., and Juvin, D. 1996. Structure from controlled motion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(5):492–504.

Coste-Manière, E., Wang, H., Rock, S., Peuch, A., Perrier, M., Rigaud, V., and Lee, M. 1996. Joint evaluation of mission programming for underwater robots. In *Proc. of IEEE International Conference on Robotics and Automation*, Minneapolis, pp. 2492–2497.

Espiau, B., Chaumette, F., and Rives, P. 1992. A new approach to visual servoing in robotics. *IEEE Trans. on Robotics and Automation*, 8(3):313–326.

Espiau, B., Kapellos, K., Jourdan, M., and Simon, D. 1995 On the validation of robotics control systems – part i: high-level specification and formal verification. INRIA Research Report 2719.

Espiau, B., Kapellos, K., Coste-Manière, E., and N. Turro, N. 1996. Formal mission specification in an open architecture. In *Proc. of ISRAM '96.*

Halbwachs, N., Lagnier, F., and Ratel, C. 1992. Programming and verifying real-time systems by means of the synchronous data-flow programming language LUSTRE. *IEEE Transactions on Software Engineering, Special issue on the Specifiction and Analysis of Real-Time Systems*, 18(9):785–793

Halbwachs, N. 1993. *Synchronous programming of reactive systems*, Kluwer, Germany.

Harel, D. and Pnueli, A. 1995. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI Series*, pages 477–498, Springer, New York.

Harel, D. 1987. STATECHARTS: a visual formalism for complex systems. In *Science of Computer programming*, 8(3):231–274.

Jourdan, M., Lagnier, F., Maraninchi, F., and Raymond, F. 1994. A multiparadigm language for reactive systems. In *Proc. of the IEEE Int. Conf. on Computer Languages*, Toulouse, France.

Kapellos, K., Abdou, S., Jourdan, J., and Espiau, B. 1995 Specification, formal verification and implementation of tasks ans missions for an autonomous vehicle. In *Proceedings of the 4th Int. Symp. on Experimental Robotics*, Stanford.

Kŏseká, J., Christensen, H., and Bajcsy, R. 1995. Discrete event modeling of visually guided behaviors. *International Journal of Computer Vision*, 14(2):179–191

Le Borgne, M., Benveniste, A., and Le Guernic, P. 1989. Polynomial ideal theoric methods in discrete event and hybrid dynamical systems. In *Proc. of 28^{t}h IEEE Conf. on Decision and Control*, Tampa, Florida, pp. 2665–2700.

Le Borgne, M., Marchand, H., Rutten, E., and Samman, M. 1996. Formal verification of SIGNAL programs: application to a power transformer station controller, in *Proc. of the 5th Int. Conf. on Algebraic Methodology and Software Technology, AMAST'96*, Munich, Germany, LNCS No 1101, Springer Verlag, pp. 270–285.

Le Guernic, P., Le Borgne, M., Gautier,T. and Le Maire, C. 1991. Programming real time application with SIGNAL, *Proceeding of the IEEE*, 79(9):1321–1336.

Marchand, E., Chaumette, F., and Rizzo, A. 1996. Using the task function approach to avoid robot joint limits and kinematic singularities in visual servoing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'96*, Osaka, Japan, pp. 1083–1090.

Marchand, E., and Chaumette, F. 1997a. Active sensor placement for complete scene reconstruction and exploration. In *IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 743–751, Albuquerque, New Mexico, April 1997.

Marchand, E., Rutten, E., and Chaumette, F. 1997b. From data-flow task to Multi-Tasking: Applying the synchronous approach to active vision in robotics *IEEE Trans. on Control Systems Technology*, 5(2):200-216.

Marchand, H. and Le Borgne, M., 1997. Partial Order Control and Optimal Control of Discrete Event Systems modeled as Polynomial Dynamical Systems over Galois Fields. *Rapport de Recherche IRISA*, No 1125, October.

Medeiros, A., Chatila, R. and Fleury, S. 1996. *Specification and validation of a control architecture for autonomous mobile robots.* IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'96), Osaka, Japon, Volume 2, pp.162-169

Ramadge, P.J., and Wonham, W.M. 1989. The control of discrete events systems. *Proceedings of the IEEE*, 77(1):81–97.

Rutten, E., and Le Guernic, P. 1995. *Sequencing and preempting data flow tasks*, Proc. of the 20th IFAC/IFIP Workshop on Real Time Programming, WRTP'95, Fort Lauderdale, Floride.

Schneider, S., Chen, V., and Pardo-Castellote, G. 1995. The ControlShell component-based real-time programming system. In *Proc. of the IEEE Int. Conf on Robotics and Automation*, Nagoya, Japan, pp. 2381–2388.

Simon, D., Espiau, B., Castillo, E. and Kapellos, K. 1993. Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Controller Issues, *IEEE Trans. on Control Systems Technology*, 1(4):213–229.

	general purpose languages	Concurent programming languages	Automata	Petri nets	Synchronous languages		
Bases							
family of language	asynchronous	asynchronous	asynchronous	asynchronous	synchronous		
language	C/F77	ADA/CSP			ESTEREL	LUSTRE	STATECHART
style of programmation	imperative	imperative	imperative		imperative	data-flow	visual
graphic interface	no	no	yes	yes	yes		yes
determinism		no	yes	not clear	yes	yes	
parallelism	no	yes	no	yes	yes	yes	
sequencing	yes	yes	yes	yes	yes	yes	
preemption	not easily	not easily	no	no	yes	yes	
Formalism for							
Continuous part	none	no	no	no	no	yes	no
Tasking part	none	yes	yes	yes	yes	yes	yes
Integration C/D	no	no	no	no	no	yes	no
Verification of							
continuous part	no	no	no	no	no	no	no
discrete event behavior	no	no	yes	yes	yes, AUTO	yes	
Adequation to							
control theory	g^{al} purpose lang.				no	yes	no
mission specification	g^{al} purpose lang.	yes	yes	yes	yes		yes
availibility	wide	wide					

Table 1: Functionalities of different formalism or languages (though SIGNAL is a synchronous language, it appears in Table 2 as an environmenent)

	KHEOPS	CONTROLSHELL	ORCCAD	SIGNAL
Bases				
family of language	synchronous	asynchronous	synchronous	synchronous
style of programmation		data-flow	imperative	data - flow
language	PRS (colored Petri nets)		ESTEREL	SIGNAL
graphic interface		yes	yes	yes
determinism			yes	yes
parallelism	yes		yes	yes
sequencing	yes	yes	yes	yes
preemption	not clear		yes	yes
Formalism for				
Continuous part	other language	data-flow	through SIMPARC	data flow language
Tasking part	Automata	finite states machine	based on automata	based on time interval
Planning part	Rule-based system	none	none	none
Integration C/D	no	yes	no	yes
Verification of				
discrete event behavior	logical and temporal	not clear	AUTO	SIGALI
continuous part			simulation	no
Level of integration	partial different models	partial different models	partial	complete (based on a unique model)
Adequation to				
control theory	no	yes	partial	dedicated
mission specification	yes	yes	dedicated	yes
availibility	Unknown	commercial	available	commercial (free for research)

Table 2: Functionalities of various CACE